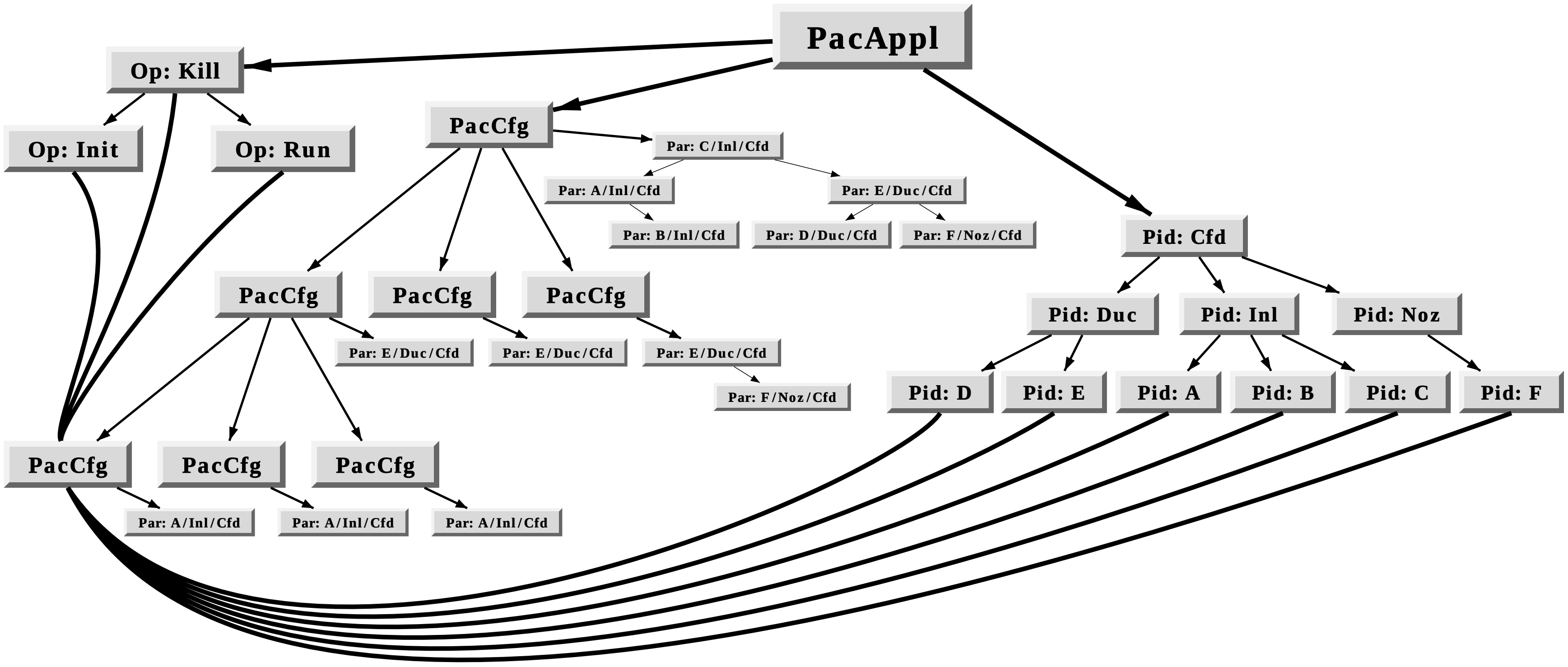
PRICE Application Wrapping

The goal is to wrap applications into a single architectural form providing for the day to day operation and use of that application while having no *a priori* knowledge of the nature of the application.

1. Wrapped applications are to be self-revealing. Consuming code is not to require foreknowledge of the nature of a particular application.
2. Wrapped applications are to persist through multiple sessions.
3. Wrapped applications are to accept evolutionary changes gracefully.
4. The strategy is to survive changes of the support team.

Web Reference: **http://www.lerc.nasa.gov/WWW/price000/index.html**
EMail: **William.H.Jones@lerc.nasa.gov**



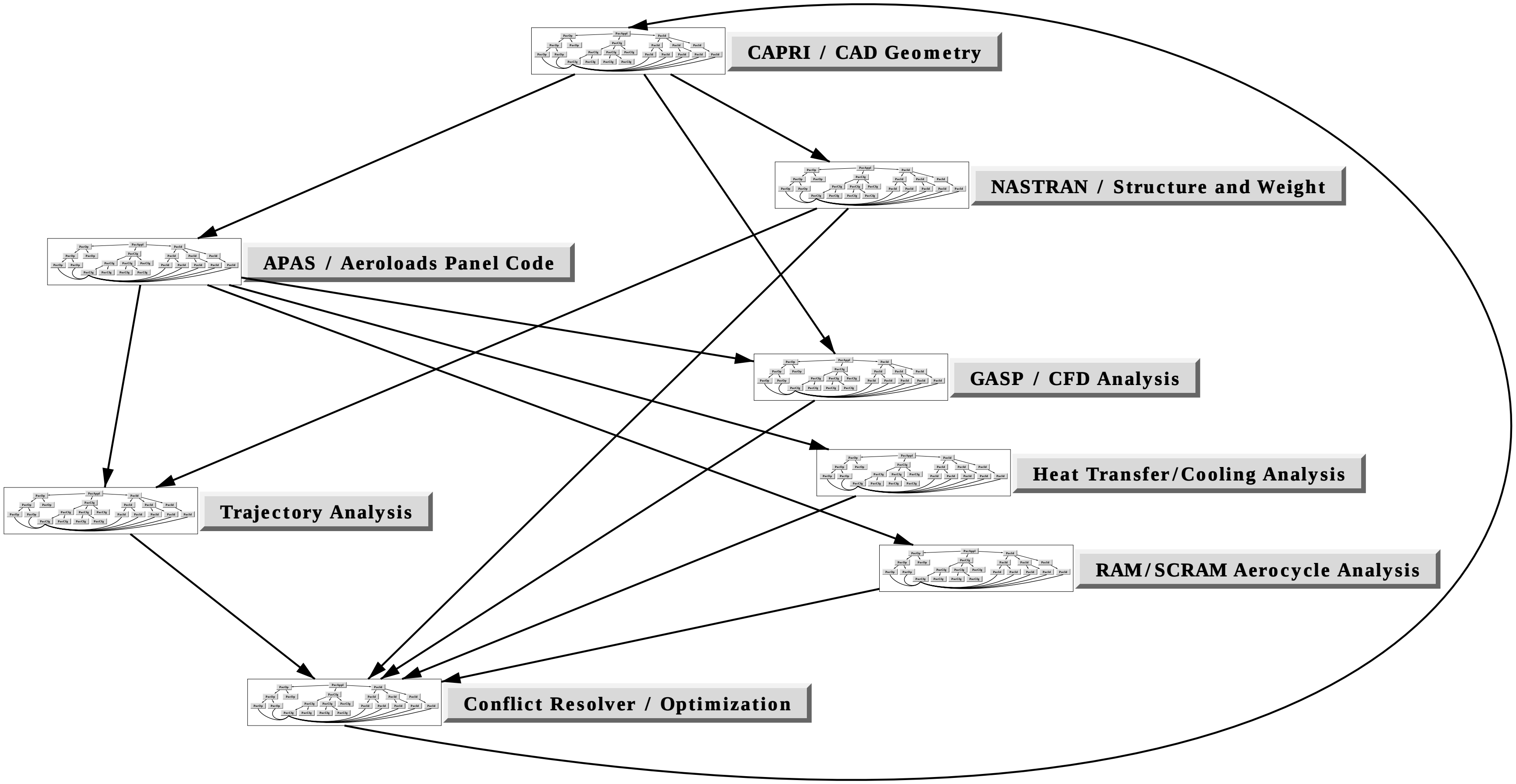
PRICE Application Structure

Key Features of the PacBObj Class

1. **PacBObj** inherits the ability to participate in a directed graph.

2. **PacBObj** implements an ability to be described. Such descriptions are layered by the derived class hierarchy and are optional at every level. Descriptions include, but are not limited to, the following.

    1. A name,
    2. An access control list,
    3. An annotation,
    4. A short blurb (a quick help text),
    5. A change history,
    6. A drop-down prompt (as in a pause of the cursor on an item),
    7. A Universal Resource Locator (URL),
    8. A descriptive (multi-line) text (as in an engineering log),
    9. A type (as in double, long, scalar, array, and the like), and
    10. A units text.



CAPRI / CAD Geometry

NASTRAN / Structure and Weight

APAS / Aeroloads Panel Code

GASP / CFD Analysis

Heat Transfer/Cooling Analysis

Trajectory Analysis

RAM/SCRAM Aerocycle Analysis

Conflict Resolver / Optimization

Approximate Rocket-Based Combined Cycle Engineering Solution Flow

Coordinated Automatic Recirculation of Design Feedback

CORBA Migration

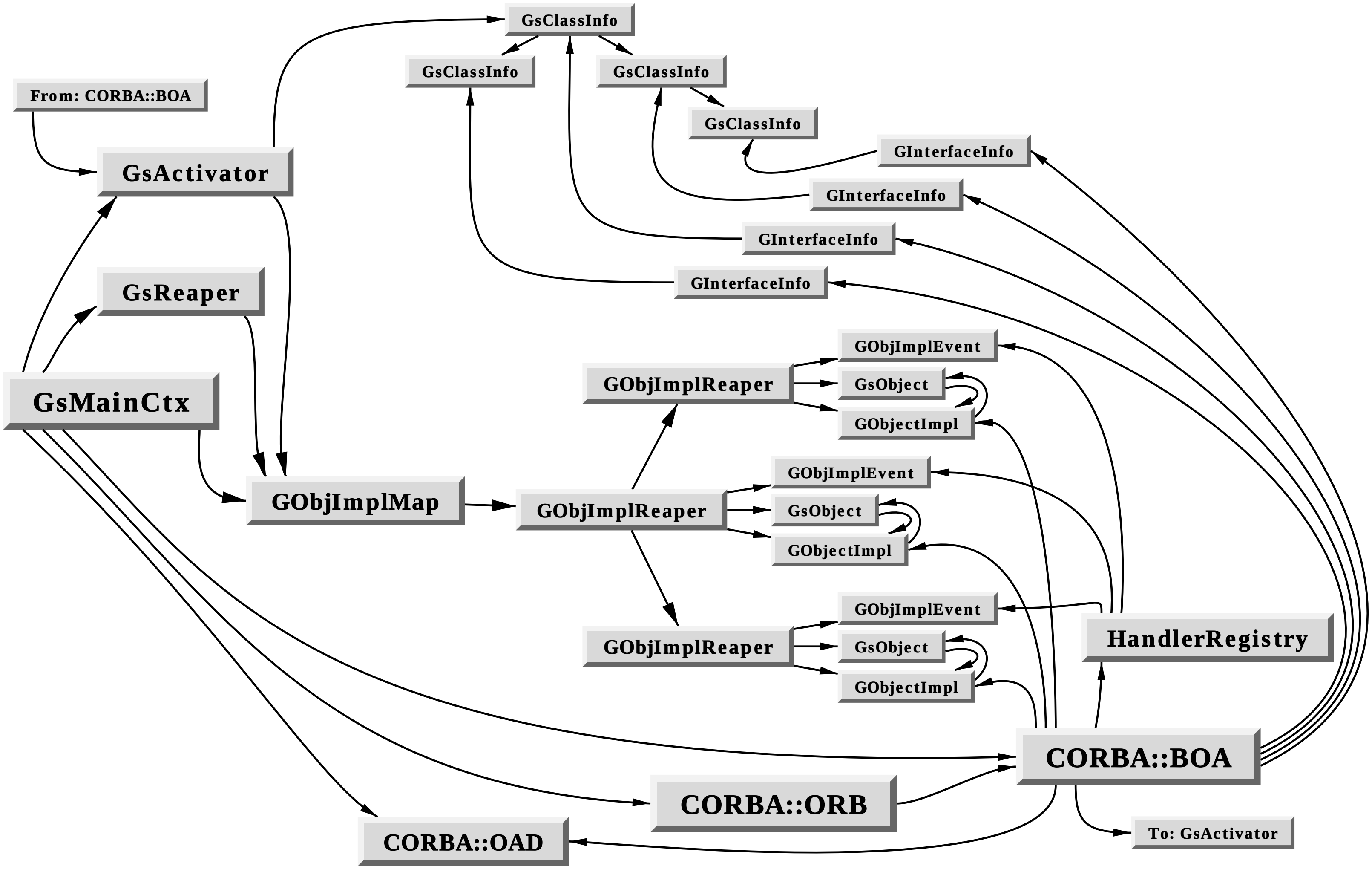
Real life requirements dictated a migration to the CORBA distributed-object environment.

1. Real applications require enormous data capacities (10 Gigabytes and up) which cannot be accommodated in a single C++ virtual address space. Service activated objects (Visigenic feature?) work around this problem.

2. A need exists to provide for multiple accessors, allowing for both generators and browsing consumers of information.

3. A cross-language capability is needed to allow consuming tools (such as GUIs) to be written in languages such as Java.

4. Some application providers wish to provide service without revealing implementing technology.


GsClassInfo
GsClassInfo
GsClassInfo
From: CORBA::BOA
GsClassInfo
GInterfaceInfo
GsActivator
GInterfaceInfo
GInterfaceInfo
GsReaper
GInterfaceInfo
GObjImplEvent
GObjImplReaper
GsObject
GsMainCtx
GObjectImpl
GObjImplEvent
GObjImplMap
GObjImplReaper
GsObject
GObjectImpl
GObjImplEvent
HandlerRegistry
GObjImplReaper
GsObject
GObjectImpl
CORBA::BOA
CORBA::ORB
CORBA::OAD
To: GsActivator

Key Elements of the GObject Interface

1. Identification by interface, instance name, type code, and supporting interface information instance is provided.

2. The interface information instance provides a self-revealing factory for all derived interfaces.

3. The ability to be ‘destroyed’ (in the manner of a C++ **delete** operator) is provided.

4. A stub-connection mechanism is implemented as a key element of service activation.

5. A rudimentary directed graph capability is provided for internal, inter-object operations.

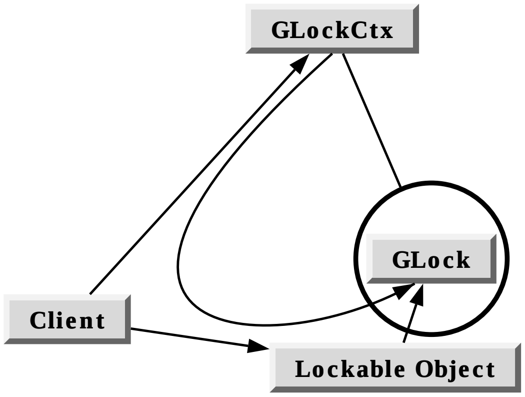
Distributed Lock Management and Deadlock Detection

A distributed lock system is based upon the following component triad.

1. The **GObjLck** interface (itself derived from the **GObject** interface) provides base functionality for things that can be locked.

2. The **GLockCtx** interface provides a nestable user/thread context within which locks are held. Such a lock context must be supplied as an argument to each method invoking lock services.

3. The **GLock** interface provides a lock which is cognizant of the lock contexts referring to it.



Release, **Reference**, **Read**, **Write**, and **Delete**, locks are supported with typical multiple-reader/single-writer protocols. Deadlock detection is done by the lock context on a per-thread basis. Deadlocks are to be typically resolved by releasing all locks and trying again ‘later’.

Positional Iterators

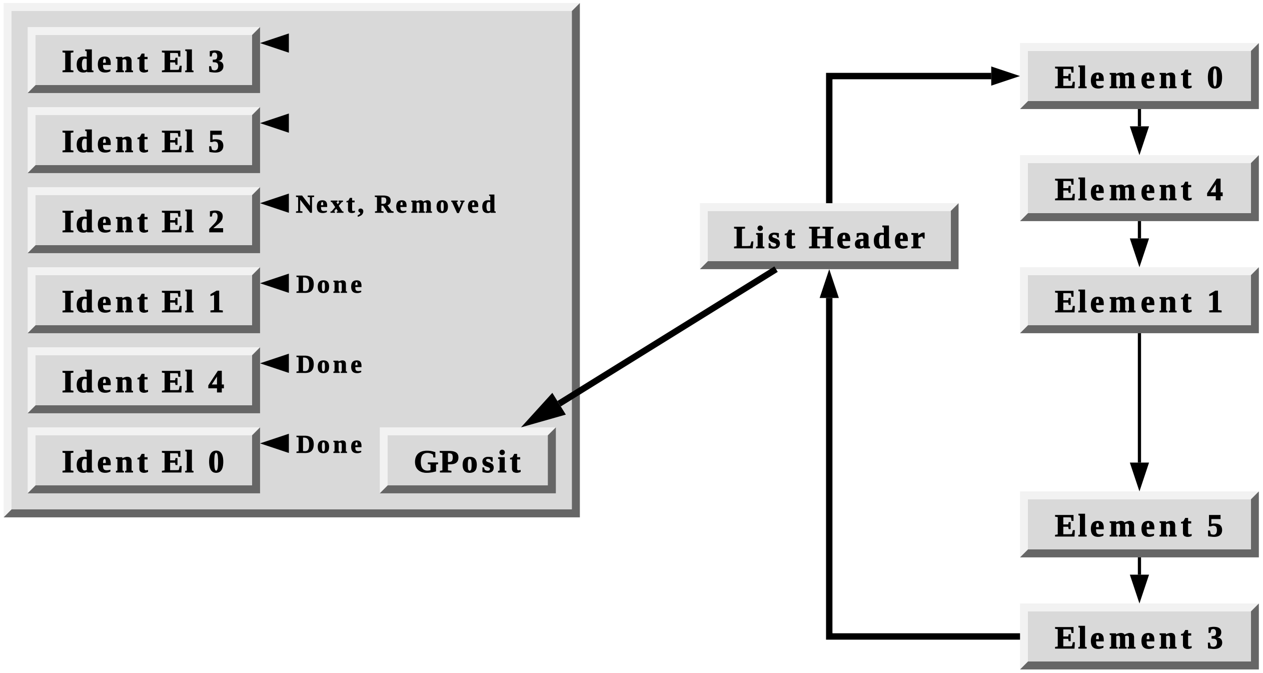
The **GPosit** interface encapsulates the mechanics of traversing an ordered set which may be changing during the time of traversal. This capacity is used by the **GLock/GLockCtx** interface pair to allow traversal of conflicting lock contexts while lock activity continues.

1. At iterator initialization, the ordered set is traversed and identifiers transferred to the iterator. For brief element visitations, this is not a highly efficient choice.

2. The initialized iterator becomes a ‘successor’ of the controlling element of the ordered set (using **GObject** base functionality).

3. Each successor iterator is informed of each subsequent addition to the set. The iterator may choose to ‘go back’ and pick up such new elements even though traversal may already be beyond that point.

4. Each successor iterator is similarly informed of deletions from the set. Iterators note such deletions, but retain the element so that, should the current position be on that element, a nasty re-positioning search will not be required.



Effect of Element Removal on an Operating Iterator

Protected Member Functions

The CORBA Interface Definition Language makes no provision for **protected** or **private** methods in the manner of C++ class member functions; however, these capabilities are needed to allow instances of an interface to manipulate internals of other instances of the same interface without exposing those internals to external consumers of the interface. For instance, the manipulation of the link elements of the **GList** interface must be protected from outside, potentially corrupting, manipulations.

1. The **GsObject** implementation of the **GObject** interface provides internal functions for the generation and validation of encrypted, digitally signed messages.

2. An implemented method wishing to invoke a protected method of the interface must generate such a message. Because the encryption keys are made available only to interface implementation code, only such code may generate valid protection messages.

3. Each protected method requires a protection message as an argument to the method. If the message does not validate, the protected method simply exits without performing the protected function.

Furthering Distributed Object Protections

It is expected that the concept of an identifiable ‘user’ will be introduced at the application derivational level. It will be desirable to restrict certain operations, for instance the creation of a new application instance, to known users.

1. User identification objects and access control lists will be applied as has already been demonstrated (but not generally applied) in the C++ implementation.

2. User identification objects may be explicitly supplied to methods, or carried along in instances of a derived form of **GLockCtx** interface. By so doing, interfaces derivationally below the application level may continue to manipulate objects even though per-user access controls have been introduced.

3. Similarly, a derived **GLock** interface form may apply user access controls to lockable objects which, themselves, are derivationally below the user conceptual level.

4. The **OperDisabled** exception allows methods to be turned off when no longer appropriate at some advanced derivational level. For instance, the zero-argument **CreateInterface** method may be disabled in a derived **GInterfaceInfo** interface to prevent unidentified users from creating applications.

Interfaces Developed to Date

GArr *et al* :: Base array functionality, arrays of various forms including arrays of interface instances.

GInterfaceInfo :: Interface support information, generic instance factory.

GList, GListGObj :: Distributed linked list base functionality and a linked list of interface instances.

GLock, GLockCtx :: Distributed lock management facilities, deadlock detection.

GMap :: Distributed balanced binary tree functionality.

GObject *et al* :: Basic service-activated, persistent distributed object foundation, objects with boolean characteristics, lockable objects.

GPosit :: Positional iteration on dynamically changing sets.